Eventual Leader Election in Infinite Arrival Message-passing System Model with Bounded Concurrency

Sara Tucci Piergiovanni, Roberto Baldoni
University of Rome “La Sapienza”
EDCC 2010 – Valencia (Spain)

We want to thank Francois Bonnet that kindly accepted to deliver this talk
Dynamic Distributed Systems: Context & Motivations

- Advent of Complex Distributed Systems (p2p, sensor networks, mobile networks etc…)
  - new problems (information dissemination, content distribution and retrieval, service orchestration and composition, location dependent computing, smart spaces etc.)
  - practical solutions on top of a system with variable size in space and time
  - lack of any formal framework
Dynamic Distributed Systems: Assessing assumptions through applications

- each process autonomously decides to locally run the same distributed application, when joining (it becomes up) and leaving the system

- it is impossible to know the set of processes participating to the computation because it could be potentially infinite

- As extremes, in some moments the system could cease its existence as no process is currently active and at some other moment the system is made of tens, or thousands, of active processes
Dynamic Distributed Systems: Assessing assumptions through applications

- The system
  - does not start with a known and pre-defined setting
  - is just the “sum” of all running entities and their local configurations

- each entity has to learn what the system is at run-time in order to successfully reach system goals
Static distributed Systems

- main characteristics: a predefined setting i.e.,
  - the application knows, directly or indirectly, the set of processes that will participate to the computation.
  - The application also knows if it can exploit synchrony assumptions

- This has a noteworthy consequence: the system can be carefully and "centrally" configured through an appropriate tuning phase in order to get the best performance.

- The application cycle is: Design, deployment, configuration, final deployment, operational

- Air traffic control, Financial systems, Aereospace systems, Egov, Telco service continuity, and many others are examples of static distributed systems
Dynamic Distributed Systems: Uncertainty in Distributed Systems

- Static Distributed Systems
  - Lack of temporal knowledge
  - failures
  - unknown communication delays

- Dynamic Distributed Systems
  - Same as in static distributed systems
  - non-monotonic and unknown size of the system (due to churn)
  - neighborhood

- Solid theoretical foundations
- Precise problem specifications
- Rigorously correct solutions
Dynamic Distributed System Model

- The number of processes in the entire computation is infinite

- At each time t the number of processes progressing in the computation is bounded but unknown

- Processes have a unique identifier given by the pair (IP_address, P_id)

- Processes are equipped with a correct clock

- Important: continuous change of membership (churn phenomenon). Processes do not notify to anyone when they join the computation and when they leave it.
Processes

- Processes can be *up* or *down*
  - Initially all the processes are down
  - Each process may become up at some arbitrary time
  - Each up process may later become down

- Only up processes may take computational steps

- The number of up processes is bounded at each time by a constant C (i.e., bounded concurrency)

- A process is *good* iff it is eventually and permanently up

- A process is *bad* iff it is eventually up and each time it is up it turns to be down within a finite time
Communication

- Processes may communicate by means of two fair lossy primitives
  - Multicast (e.g., IP multicast)
  - Unicast

- Assumptions:
  - There exists a finite and unknown bound $L_{\text{max}}$ such that the number of consecutive message losses on a link does not exceed $L_{\text{max}}$.
  - There exists a finite and unknown bound $\delta_{\text{max}}$ such that the message transfer delay does not exceed $\delta_{\text{max}}$.

- In other words, the period a good process does not get messages from another good process is bounded
Dynamic System Issues

- No initial knowledge of the set of processes that will be part of the system
  - good processes are not known in advance
- Heartbeat may arrive both from good and bad processes

- Dealing with a possibly infinite set of not correct processes that may wake up at any time
Eventual Leader Election solution ($\Omega$)

- Each Process sends Heartbeats to all processes in the system through the multicast primitive

- Each process receiving heartbeats from other processes forms a list of alive processes

- A leader is elected from this list using a deterministic rule (the same at each process e.g., the process with the lowest identifier)

- Target: all processes have to eventually elect the same good process
Solutions applying to Crash failure model do not work

- Heartbeats arrives from good and bad processes over the entire computation

- The alive list at each process continuously will include good and bad processes

- Risk: elect an infinite sequence of bad processes
Solutions applying to Crash-recovery failure model do not work

- Use of epochs to distinguish different lives of a process. Such epochs are stored in a stable storage.

- The epoch number of a good process eventually stabilizes to a constant number.

- The leader is thus selected from the alive list as the one with the lowest identifier among the ones with the lowest epoch number.

- Our model does not include stable storage.
Ω in Dynamic Distributed System is done in 2 Steps

- The HB* Oracle
  - Provide a list of processes deemed to be up (alive list). The list aims to:
    - Put good processes on the top of the list
    - Stabilize the position of a good process in the list

- Ω protocol
  - Take the list provided by the HB* protocol and output the leader
The HB* Oracle

- Provide a list of processes deemed to be up (alive list)

**Specification**

- **Completeness**: at each good process $p_i$, the alive list will eventually include all good processes permanently.

- **Accuracy**: Eventually and permanently, at each good process $p_i$, for each good process $p_j$ in the alive list, each good process $p_k$ ordered before $p_j$ is a good process.

- **Stability**: eventually and permanently, at each good process $p_i$, for each good process $p_j$ in the alive list, $p_j$ will occupy a fixed position in the list.
HB* Implementation

- Each process periodically sends an heartbeat with its age (a local sequence number)
- Each process locally sorts alive processes according with their age

Assumptions:
- All processes grow at the same frequency
- FIFO order of heartbeat is never violated
- There are unknown bounds on message losses and delays

Completeness directly follows
HB* Accuracy

- counting heartbeat is not sufficient to ensure accuracy

- An infinite sequence of bad processes can always be on the top of the Alive list above a good process (due to lost message and infinite arrivals)

- communication assumptions ensure this can happen only for a bounded period of time
HB* Stability

- counting heartbeat is not sufficient to ensure stability

- The two processes change infinitely often their positions in the HB* list

- assign life_bonus to increment artificially the age.
Ω Implementation based on HB*

- (Eventual Leader). There exists a time $t$ after which every good process elects the same good process.

- Processes acquire the alive list from the HB* oracle and trust one good process in its list.

- Note that despite HB* ensure stability, accuracy completeness of the alive list, the following situations might happen:
  - alive lists may always contain bad processes and
  - different alive lists may never reach the same order on good processes

- Assumption. The number of good processes is lower bounded by a known constant $b$ such that

$$b \geq \lfloor \frac{C}{2} + 1 \rfloor$$
Implementation based on HB*

Step 0. first “b positions” of the Alive lists are sent, trusted list, by a process to each member of its alive list.

Each time a trusted list is received by a process p from process q:
1. If q is in the $k \leq b$ positions of the trusted list of p then
   1. the process computes the new local trusted list as the union between its current local trusted list and the one just received.
   2. If the new trusted list is different from the previous one the trusted list is sent to all the processes in the alive list.

The majority assumption implies that at least one good process will be eventually included in the trusted list.

A candidate list is obtained at each process p from the intersection of the trusted lists received by processes belonging to the first b positions of HB*.

The leader is the process with the minimum identifier in the candidate list.
Conclusion

- Dynamic Distributed Systems are unquestionably more complex than static ones. This leads to more complex solutions to solve the same problem.

- We addressed the problem of Eventual leader election.

- Our solution introduces a new failure detector HB* which has stronger properties with respect to classical FDs for static distributed systems (i.e., process stability).

- Interestingly, our solution applies to a crash-recovery model where processes do not have access to stable storage.
Thank You!

Questions!?